ROBUST PATH PLANNING WITH IMPERFECT MAPS

Dave Ferguson* and Anthony Stentz
Robotics Institute, Carnegie Mellon University
Pittsburgh, PA, 15213

ABSTRACT

We describe an efficient method for path planning in environments for which prior maps are plagued with uncertainty. Our approach processes the map to determine key areas whose uncertainty is crucial to the planning task. It then incorporates the uncertainty associated with these areas using the recently developed PAO* algorithm to produce a fast, robust solution to the original planning task. We present results from a simulated outdoor navigation scenario.

1 INTRODUCTION

Unmanned Ground Vehicles (UGV's) have the potential to be integral components of the future Army. Their ability to autonomously traverse rugged terrain makes them ideal candidates for several key tasks, such as reconnaissance and supply runs. A vital aspect of this autonomy is the generation of robust paths for the vehicle to navigate.

Much progress has been made in the field of path planning for outdoor navigation. As a result, we have very effective algorithms for providing paths from one point to another while considering a range of factors, including: distance traveled, energy consumed, visibility, time taken, and intelligence gained. Further, extensions to these algorithms have been developed that allow for onboard replanning of paths when information is received through sensors that conflicts with the original information used for planning (Stentz, 1995). These extensions are important since the initial map information held by a ground vehicle is rarely perfect.

In this paper, we address the problem of planning paths through environments for which prior maps are plagued with uncertainty. Our approach processes the map to determine key areas whose uncertainty is crucial to the planning task. It then incorporates the uncertainty associated with these areas using the recently developed PAO* algorithm to produce a fast, robust solution to the original planning task.

We begin by discussing the nature of the problem and describe current approaches to planning with uncertainty. We go on to introduce our novel solution and provide key results and extensions.

2 PLANNING WITH UNCERTAINTY

Consider a UGV navigating outdoors with a low-resolution overhead map generated by a helicopter or satellite. This map may be quite inaccurate: the density of the original data may be low and the position estimation used to project this data onto the overhead map could contain significant error. As a result, any planning map extracted from this overhead map and used by the UGV will be imperfect. In particular, when a grid-based representation of the environment is used, the terrain associated with each grid cell may be only partially known.

Figure 1 shows an artificial sample uncertainty distribution over the terrain associated with a particular cell in such an environment. Although we may not have perfect information regarding a cell's terrain, we can often extract such terrain distributions using the information we do have and some error models. In other words, by paying close attention to the uncertainty associated with our information, we can derive distributions over the possible terrain values of a given area of the environment.

Dealing with distributions over terrains rather than fixed values requires some modification to classical planning techniques. Currently, there are two common methods of planning that incorporate this type of uncertainty.

The first method, known as **assumptive planning** (Nourbakhsh and Genesereth, 1996), computes an approximate cost of traversing each cell (or 'assumes' a default value) and plans using these approximations. When the resulting plan is executed, if a discrepancy is found between a cell's assumed cost and actual cost, the plan can be updated to reflect the newly acquired information. This type of planning is very fast, since it can use A* techniques to focus its initial computation and D* techniques to repair previous plans (Nilsson, 1980; Stentz, 1995; Koenig and Likhachev, 2002).

However, planning with approximate costs breaks

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
00 DEC 2004	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Robust Path Planning With Imperfect Maps	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Robotics Institute, Carnegie Mellon University Pittsburgh, PA, 15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM001736, Proceedings for the Army Science Conference (24th) Held on 29 November - 2 December 2005 in Orlando, Florida. , The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**UU**	**7**	

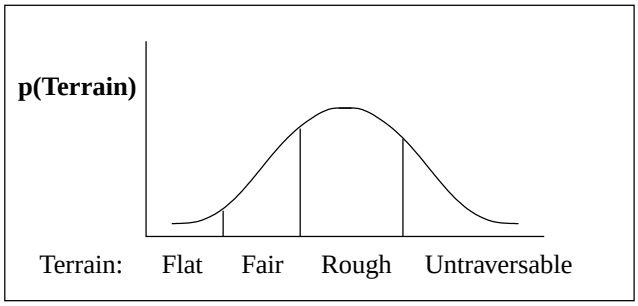

Figure 1: A sample terrain distribution

down when cells have some probability of being untraversable. Typically, assumptive planners solve for a path from the start state s to a goal state g while minimizing the overall cost of the path. The minimum cost from a cell x to the goal g is defined as:

$$Cost(x, g) = \min_{y \in nbrs(x)} [(Cost(x, y) + Cost(y, g)].$$

The value $Cost(x, y)$ is computed from the approximate cost of traversing from cell x to neighboring cell y. But if there is some non-zero probability that cell x is untraversable, the equation above is no longer valid, as it does not account for this possibility (and its associated cost). In order to get the appropriate cost, an alternative path needs to be planned *around* x to the goal. The cost of this path is nontrivial to compute, as it requires making similar considerations for all other cells encountered, which makes the overall computation exponential in the number of cells in the environment. As a result, accurate cost approximations are difficult to generate and computed solution paths based on rough estimates can be highly sub-optimal.

A second method addresses the limitations of approximating costs and deals with terrain uncertainty more comprehensively. **Partially observable planning** (Kaelbling et al., 1998) generates every possible terrain of every cell and finds the overall best plan taking into account these possibilities and their associated probabilities. Thus, rather than using an approximation of the cost of using a particular cell on a path to the goal, it computes the true expected cost by generating every possible outcome and its associated cost and probability. However, as mentioned above, computing these costs is exponential in the number of cells in the environment. Moreover, as an agent moves through its environment it learns the true terrain values of areas it encounters. Consequently, the terrain distribution associated with each cell can change and this change needs to be taken into account in the planning stage. As a result, partially observable planning is often intractable for reasonably sized environments. Even if our terrain model is reduced to two possibilities, traversable and untraversable, and the agent is equipped with a perfect contact sensor (the simplest case), planning involves dealing with $3^{m \cdot n}$ information states[1] if the planning grid is $m \cdot n$ cells in size. This is a prohibitively large state space.

1. Initially, set the cost of each cell in the environment to its expected traversable cost. Use D* to plan an optimal path (relative to these costs) from the robot position to the goal. Mark the pinch points along this path (see 2(b)) and use them to construct an ordered list P.
2. While there are still pinch points in P:
   (a) *Replan path from pinch point to goal*: Remove the top pinch point from P and set the terrain of each cell belonging to a pinch point to *untraversable*. Invoke D* to replan a path to the goal from the cell previous to the pinch point (on the path on which the pinch point was found).
   (b) *Mark new pinch points along path*: Step along this new path and look for sections that have a high probability of being untraversable *and* are costly to navigate around. Mark these cells as pinch points and add them to the end of P.
3. Return all pinch points encountered.

Figure 2: Pinch Point Extraction Algorithm

However, it is possible to restrict our attention to areas of the state space that are likely to be important to an agent navigating the environment. In this way, we gain the significant advantage of partially observable planning while incurring only a fraction of its computational cost.

3 EXTRACTING PINCH POINTS

The idea is to focus our computation on areas of the environment whose traversability, and hence uncertainty, is crucial to the planning task. Thus, rather than dealing comprehensively with the uncertainty associated with every cell, as partially observable planning does, we restrict our attention to those cells that are most useful.

These are the ones that would cause a costly detour in the robot's path to the goal if they turned out to be untraversable. We would like to be able to detect these cells and incorporate their terrain uncertainty in our planner so that we could decide from the outset whether it is better to avoid these cells entirely or risk going through them.

[1]Each cell may be known to be traversable (t), known to be untraversable (o), or not yet seen by the agent (u), in which case it has its initial probability distribution over being traversable/untraversable. This results in 3 different states of *information* the agent may have concerning the terrain of each cell.

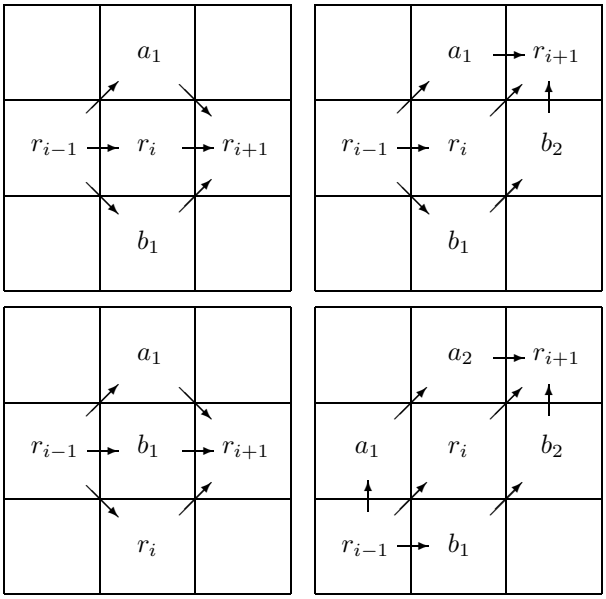

Figure 3: Finding potential path blockages in an eight-connected grid. Each diagram highlights three local paths from cell r_{i-1} to r_{i+1} for different relative positions of cells r_{i-1}, r_i, and r_{i+1}. If there is a nontrivial probability that all of these three paths are untraversable, the cells along the paths are grouped together as a potential pinch point.

To find these cells, coined 'pinch points' (Ferguson et al., 2004), we generate a set of cells that could reasonably be encountered by an agent navigating to the goal and look for key members of this set. Figure 2 outlines the process.

We first assume each cell with a reasonable probability of being traversable is in fact traversable and generate its resulting expected terrain cost[2]. We then calculate a path to the goal that is optimal relative to these costs. Given this path, labelled as consecutive cells $r_1 \dots r_n$, we can find all sections of the path that are potential blockages *and* would cause significant detours if found to be blocked.

One approximation method for finding such blockages is as follows. Label a section of the path centered on cell r_i a potential blockage if there is a nontrivial probability that an agent at cell r_{i-1} will not be able to get to cell r_{i+1} using one of the three shortest non-intersecting routes between the two cells. Here, the idea is that if r_i turns out to be untraversable, there is still a good chance r_{i+1} is reachable from r_{i-1} using one or more of the neighbors of r_i, but if r_i is untraversable and the next obvious two paths from r_{i-1} to r_{i+1} are also untraversable, then it is not so likely. To generate the probability that the path is blocked at cell r_i, we look at the three shortest paths from cell r_{i-1} to cell r_{i+1} and use the terrain distributions of the cells along these paths to determine the probability that all three paths are untraversable. If this probability is greater than some threshold, we group cell r_i and the cells along the shortest paths together as a potential pinch point with position r_i.

Figure 3 illustrates the shortest paths between r_{i-1} and r_{i+1} for four different relative positions of the consecutive path cells r_{i-1}, r_i, and r_{i+1}. The paths for all other possible relative positionings can be obtained from these four. In this figure, r_i provides one path between the two cells, while the other paths are denoted by the cells marked a_1, a_2 and b_1, b_2, respectively.

If a group of cells is marked as a potential pinch point, we know that there is a nontrivial probability that an agent may not be able to get through this area. In order to determine the consequences of this possible outcome, we then check how costly a route around the potential pinch point would be. To do this, we generate a cheapest cost path from the previous cell on the path, r_{i-1}, to the next cell on the path, r_{i+1}, without using any of the cells constituting the potential pinch point. If the cost of this path is significant we add the potential pinch point to our list of true pinch points.

If we come across a number of pinch points in a row, for instance in a narrow valley, we combine them into a single pinch point. The probability of the combined pinch point being blocked is taken to be $1 - p$, where p is the probability that all of the pinch points are traversable. The position of the pinch point used later for planning is taken as the mean of its constituents.

Once we have found all pinch points along the original path, we then set the terrains of all cells comprising these pinch points to untraversable and replan paths from the close side of each of these pinch points to the goal, i.e., from r_{i-1} for each pinch point r_i. This enables us to locate new pinch points that might be encountered by the agent if it found the current pinch point to be untraversable. We iterate the procedure to generate a set of pinch points that could reasonably be encountered by an agent moving towards the goal (assuming the agent always acts optimally given its map information). Figure 4 illustrates the approach in action.

In extracting pinch points, D* is used to replan each subsequent path. The efficiency of D* over A* has been widely recognized (see (Stentz, 1995; Koenig and Likhachev, 2002)) and, as shown in the results section, this efficiency allows us to generate our set of pinch points very quickly. Once equipped with this set, we can then incorporate its members and their associated terrain uncertainties into the planning process. To do this, we make use of the recently developed PAO* algorithm (Ferguson et al., 2004).

[2]This is computed by normalising its terrain distribution to only contain the traversable range then taking an expectation.

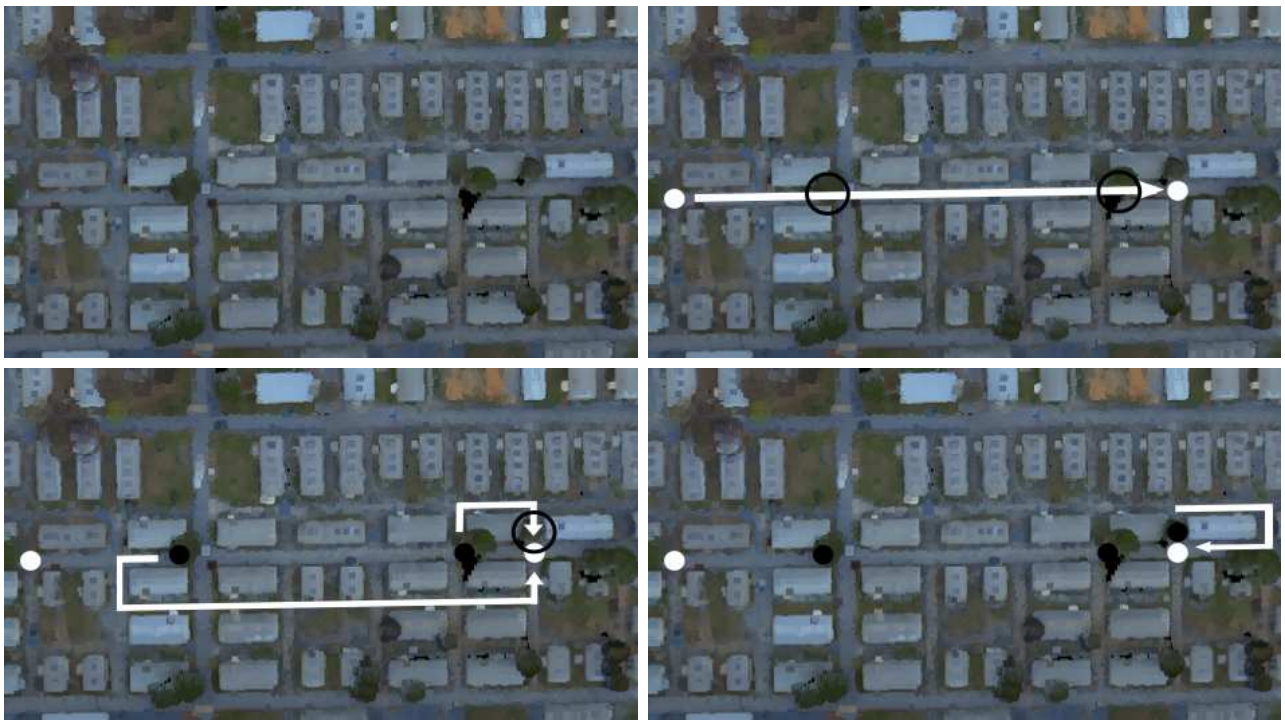

Figure 4: (*top-left*) A helicopter-generated map of an outdoor area. Dark areas represent trees or grass, light areas represent roads or buildings. Registered with this visual imagery is 3D terrain information. Black areas represent sections of the environment for which no information was gathered. (*top-right*) An optimal path (in white) from the white circle on the left side to the white circle on the right side of this map, assuming areas with a reasonable probability of being traversable are traversable. The two hollow black circles represent the pinch points found along this path. (*lower-left*) Alternative paths from the start-side of each pinch point to the goal. These paths are computed assuming the two pinch points from (*top-right*) are untraversable (denoted as black filled-in circles). A third pinch point is found along the alternative path from the second pinch point. (*lower-right*) The alternative path from the third pinch point. Since this path does not contain any further pinch points, the three black filled-in circles represent all the pinch points used for planning. Data courtesy of Omead Amidi and Ryan Miller.

4 PLANNING WITH PINCH POINTS

In (Ferguson et al., 2004) the algorithm PAO* was introduced, which solves planning problems involving hidden state such as pinch points. PAO* is applied to an adjacency graph containing the robot position, the goal position, and the pinch points in the environment. It uses the adjacency information to calculate an optimal solution graph, as with the AO* algorithm, in a highly efficient manner.

4.1 The Adjacency Graph

Each pinch point may provide a bridge between several different regions of the environment. For instance, a pinch point located at a Y-junction connects three different regions to each other. The collection of cells adjacent to the pinch point in each region constitute a *face* of the pinch point. The adjacency graph links up these faces by inserting arcs between every pair of faces that are reachable from one another. The cost of an arc between two faces represents the lowest cost associated with moving along a pinch-point free path between the faces and is used to propagate values from one face to another.

4.2 Planning with the Adjacency Graph: PAO*

A shortest path is planned from the robot position to the goal position using this adjacency graph. To do this, the problem is phrased as a search over an AND-OR graph (Rich and Knight, 1992). An AND-OR graph contains two types of nodes: AND nodes obtain their values from combining the values of all their child nodes, while OR nodes compute their values from choosing a single child node value.

Our planning domain can be represented as an AND-OR graph as follows. Each node in the graph corresponds to a face in a particular *information state*. In our setting, an information state is the state of knowledge the agent may have concerning the terrain values of each of the pinch points. Following our discussion on partially observable planning, we restrict each pinch point to be known to be traversable (t), known to be untraversable (o), or not yet seen by the agent (u).

The root of the AND-OR graph (an OR node) is the start cell s in the information state characterised by every pinch point being as yet unseen (i.e., of value u). The next

level of the graph corresponds to all elements of the adjacency graph, both faces and goal, which have arcs to s. The faces are AND nodes: each has two children representing the two possible information states realizable from visiting the node. These two children each have the same face as their parent but reside in different information states (one has the pinch point associated with the face of value *t*, the other *o*). These children are OR nodes because their associated pinch point has a known value.

PAO*, short for *Propagating AO**, is an algorithm which searches an AND-OR graph by gradually building a solution graph from the start state through two alternating phases, as with AO* (Chang and Slagle, 1971; Rich and Knight, 1992). First, it grows the best partial solution by expanding one of the non-terminal leaf nodes and assigning admissible heuristic costs to its children. Next, it uses the newly computed costs to propagate cost revisions throughout the partial solution graph. At each stage in this propagation, OR nodes which are part of the current partial solution update their choice of child to reflect the most recent cost values. An example partial solution graph for our domain is shown in Figure 5.

In the first phase, an initial heuristic cost for a leaf node l is obtained by solving for the cost of the 'heuristic counterpart' of l: the fully-known state characterized by the most desirable true values the pinch points in l could have. Pinch points with known values are left untouched. Pinch points not yet seen (with value u) are assigned the value t. The resulting cost is guaranteed to be admissible.

The major benefit of PAO* lies in its second phase, which involves the propagation of cost revisions. Unlike AO*, PAO* propagates cost changes not only upwards to parents in the partial solution graph, but sideways to neighbors (in the complete AND-OR graph), and downwards to children. The resulting approach makes full use of all received information and thus allows for more informed decisions to be made at each stage of the process. The complete algorithm is given in Figure 6 and thoroughly discussed in (Ferguson et al., 2004). Briefly, there are three key propagation steps that PAO* performs but AO* does not.

Firstly, when the cost of a face in an information state changes, PAO* propagates this updated cost *across* all faces in the information state, so that dependent faces will have their costs updated. While AO* only propagates information up the partial solution graph, PAO* also propagates it across the full AND-OR graph at each level.

Secondly, PAO* uses the nature of the current problem domain to propagate cost changes *down* the AND-OR graph. Given an AND node with two children corresponding to the two possible true values of the node's pinch point (traversable and untraversable), the cost of the parent node

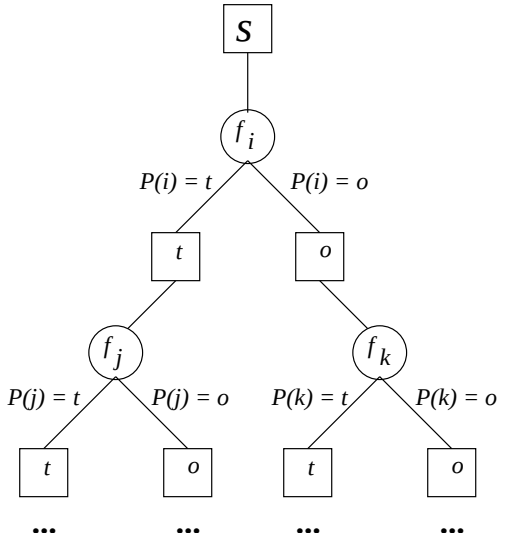

Figure 5: An example partial solution graph. Each circle corresponds to an AND node and each square to an OR node. *P(i) = o* represents the probability that the pinch point associated with face *i* is untraversable.

should never be greater than the cost of the untraversable child node. Taking advantage of this piece of intuition, PAO* updates the face costs of the states associated with untraversable nodes so that they are lower bounded by their parent state values. This also enables PAO* to provide a more realistic value for the untraversable child of each newly expanded AND node.

Similarly, the cost of a parent node should never be less than the cost of its traversable child node. Thus, PAO* also updates the face costs of parent information states so that they are lower bounded by their traversable child states.

These propagation steps combine to allow information gained at one end of the solution graph to be accessible at the other. As a result of these differences, PAO* has been shown to be orders of magnitude more efficient than AO* while still guaranteeing optimal solutions (Ferguson et al., 2004).

5 RESULTS

Focussing our computation on pinch points, we are able to solve the planning problem much more efficiently than the full partially observable approach yet still incorporate key areas of uncertainty to produce robust paths. In addition, the complexity of our approach, through both the extraction of pinch points and the subsequent PAO* search for a solution, is highly dependent on both the quality of the information the planning agent holds and the nature of the environment. Thus, when confronted with simple environments and reliable information, it is able to exploit these desirable attributes to produce solutions very quickly.

1. The initial solution graph consists solely of the start node s in the original information state.

2. While the solution graph has some nonterminal leaf node:

   (a) *Generate fringe node*: Starting from the root, traverse down the solution graph until a nonterminal leaf node is encountered. Along the way, update untraversable child states to have their face costs lower bounded by their parent states.

   (b) *Expand best partial solution*: Expand the nonterminal leaf node and compute cost values for the information states of its children. Traversable child states are given heuristic costs. Untraversable child states inherit their parents' cost values as lower bounds then perform limited value iterations over their heuristic counterparts to potentially increase these values. Add the children to the solution graph, noting whether they are terminal.

   (c) *Propagate cost changes and update solution*: Compute an updated cost of the original leaf node given the costs of its children. If the node's cost has changed, update the cost estimates for its *entire information state* and update its parent's cost to reflect these changes. If the parent is an OR node, the current node may be replaced if it no longer provides the minimum cost. If the node is a traversable child, update the costs associated with the entire parent state to be lower bounded by the current state. Continue propagating up the graph until a node is reached whose cost does not change.

3. Return the optimal solution graph.

Figure 6: The PAO* algorithm

Meanwhile, in more complex environments with less reliable information, our approach incorporates more areas of uncertainty to produce very robust solutions.

To test the computation required to extract pinch points in environments of varying complexity, we generated 1000 grid environments, each of size 200×200. The terrain distribution of each grid cell was fractally-generated to simulate similar structure to that found in outdoor regions. These environments ranged from very open, easy to navigate areas to very complex, cluttered areas[3]. With each map, we extracted a maximum of 10 pinch points, so that our results could easily be interpreted in conjunction with the planning results given in (Ferguson et al., 2004) (where the pinch points were manually specified). As Figure 4 shows, this still enables us to consider a large number of diverse paths when the environment is highly cluttered.

Over our 1000 terrain test cases, the average number of extracted pinch points was 2.47 and the average CPU time taken for this extraction was 0.07 seconds when run on a P3 1.4 GHz processor. The maximum amount of time required to extract the pinch points from any map was 0.34 seconds (with the minimum being 0.003).

Combining these results with the computation required to plan with extracted pinch points (from (Ferguson et al., 2004)), we have the entire process conservatively taking 6.7 seconds. This time reflects the worst results reported here and in (Ferguson et al., 2004) for environments with 10 pinch points.

A complete partially observable solution over such an environment would need to contend with $3^{200 \cdot 200}$ states rather than the 3^{10} used in our PAO* planning. This is currently far too large a problem to be solved optimally. However, by restricting our attention to the most important areas of the environment, we have been able to obtain robust, realistic paths for minimal computation. The resulting approach is fast enough to be used by real systems operating with imperfect information in real environments.

6 REPLANNING

The approach described above allows us to deal with the uncertainty in our planning grid which is likely to be most relevant. But there are two settings in which this approach alone will not be sufficient. The first is when an agent finds itself in a highly unlikely situation where a large number of non-pinch point cells turn out to be untraversable and, as a result, the agent is unable to traverse its computed path. The second is when the environment is dynamic.

However, we can deal with both of these settings by augmenting our approach to perform dynamic replanning while the agent navigates the environment, as follows.

First, the agent extracts the pinch points in the environment and computes its solution path to the goal as previously described.

Next, the agent computes which neighboring cell it should move to first, based on its neighbors' costs to each reachable face and the costs from each reachable face as returned by the solution path. Initially, this is trivial: the starting cell is part of the adjacency graph so the agent knows which face to move towards. It only needs to look for the

[3]For details of the fractal generation process, see (Stentz, 1995). We used a gain of 20 and varied the number of levels from 5 to 9.

neighbor which will take it to that face with least overall cost. However, as the robot traverses towards the desired face, it is able to update the terrain information concerning encountered cells. As a result, the face that is initially least costly may not remain so. For example, if the agent finds that a number of cells along its path to the desired face are untraversable, it might not be able to reach the face without a costly diversion. In such a situation it may be less costly to move towards a different face which was initially more expensive.

We can allow for this type of online replanning by interleaving D* and PAO*. First, we use D* to maintain cheapest cost paths to every reachable face from the agent cell. Thus, each time the agent updates the terrain of an encountered cell, the new terrain cost is propagated through these paths and the path costs to each face are updated accordingly. Given these costs (and the costs from each face returned by PAO*), the agent can update the best overall face to move towards.

However, PAO* derives its efficiency by ignoring much of the AND-OR graph and only considering promising faces. Thus, only the faces along the final solution graph are guaranteed to have their optimal costs; the rest may only have admissible costs. This means we need to invoke PAO* with the current desired face each time we find we are moving towards a face which has some nonterminal leaf node in its solution graph. Thus, we take the path costs returned by D* and use these to update the arcs between the robot position and the rest of the adjacency graph. We can then use PAO* to compute the best reachable face to move towards. Because PAO* does not terminate until the solution graph is complete, we are guaranteed that the face returned by the algorithm will have its optimal cost given the current adjacency graph information. When the agent reaches a face, it can update the adjacency graph for all altered arc costs and replan accordingly.

7 Conclusion

We have presented an algorithm for efficient path planning with imperfect map information. Our approach highlights areas of the environment whose uncertainty is most important to the planning task. It then makes use of the recently developed PAO* algorithm to incorporate the uncertainty associated with these areas into the planning process. The resulting solution is optimal with respect to the extracted pinch points and their probabilities of being traversable, and is computed in a fraction of the time required by a complete solution. We have also described a replanning extension to the algorithm to cope with dynamic environments.

8 Acknowledgments

The authors would like to thank Omead Amidi and Ryan Miller for the helicopter-generated outdoor terrain data. This work was sponsored by the U.S. Army Research Laboratory, under contract “Robotics Collaborative Technology Alliance” (contract number DAAD19-01-2-0012). The views and conclusions contained in this document are those of the authors and do not represent the official policies or endorsements of the U.S. Government.

References

Chang, C. and Slagle, J.: 1971, An admissible and optimal algorithm for searching AND-OR graphs. *Artificial Intelligence* **2**, 117

Ferguson, D., Stentz, A., and Thrun, S.: 2004, PAO* for Planning with Hidden State. in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, New Orleans, LA

Kaelbling, L., Littman, M., and Cassandra, A.: 1998, Planning and acting in partially observable stochastic domains. *Artificial Intelligence*

Koenig, S. and Likhachev, M.: 2002, Incremental A*. in *Advances in Neural Information Processing Systems*, MIT Press

Nilsson, N.: 1980, *Principles of Artificial Intelligence*, Tioga Publishing Company

Nourbakhsh, I. and Genesereth, M.: 1996, Assumptive Planning and Execution: a Simple, Working Robot Architecture. *Autonomous Robots Journal* **3(1)**, 49

Rich, E. and Knight, K.: 1992, *Artificial Intelligence*, McGraw-Hill

Stentz, A.: 1995, The Focussed D* Algorithm for Real-Time Replanning. in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*